**What is the diamond problem in inheritance?**

In case of multiple inheritance, suppose class A has two subclasses B and C, and a class D has two super classes B and C.If a method present in A is overridden by both B and C but not by D then from which class D will inherit that method B or C? This problem is known as diamond problem.

#### **Advantages of making an immutable class ?**

* Immutable objects are thread safe and doesn't have synchronization issues.
* Act as keys for **Map** and values of **Set**
* final class cannot have sub class
* Another important benefit of Immutable objects is reusability, you can cache Immutable object and reuse them, much like String literals and Integers. You can use static factory methods to provide methods like valueOf(), which can return an existing Immutable object from cache, instead of creating a new one.

### Can we overload final method?

Yes. We can overload final method but can’t override them.

### What are the sequence of execution of instance block,static block and constructor?

First static block get executed while loading class. Each instance block get placed at the top of constructor method and constructor method get executed.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | package com.startwithjava;  class X{  static{  System.out.println("X static block");  }  {  System.out.println("Instance block 1");  }  public X(int x){    System.out.println("X constructor()");  }    {   System.out.println("Instance block 2");  }    }    public class ConstructorOverloading {  public static void main(String[] args) {   X x= new X(10);   }  } |

Output

|  |  |
| --- | --- |
| 1  2  3  4 | X static block  Instance block 1  Instance block 2  X constructor() |

## Class Loaders :

Class loaders are responsible for**loading Java classes during runtime dynamically to the JVM** (Java Virtual Machine).  Also, they are part of the JRE (Java Runtime Environment).Normally these Java classes aren’t loaded into memory all at once, but when required by an application. This is where class loaders come into the picture. They are responsible for loading classes into memory.

Java class loaders can be broadly classified into below categories:

* Bootstrap Class Loader  
  Bootstrap class loader loads java’s core classes like java.lang, java.util etc. These are classes that are part of java runtime environment. Bootstrap class loader is native implementation and so they may differ across different JVMs.
* Extensions Class Loader  
  JAVA\_HOME/jre/lib/ext contains jar packages that are extensions of standard core java classes. Extensions class loader loads classes from this ext folder. Using the system environment propery java.ext.dirs you can add ‘ext’ folders and jar files to be loaded using extensions class loader.
* System Class Loader  
  Java classes that are available in the java classpath are loaded using System class loader.

# [**What is the use of Custom Class Loader**](https://stackoverflow.com/questions/10828863/what-is-the-use-of-custom-class-loader)

* **Load classes from anywhere** Classes can be loaded from anywhere, **for ex, Database, Networks, or even define it on the fly**.
* Loading multiple versions of the same class with different classloaders (e.g. to resolve possible versioning conficts for example)

Java default ClassLoader can load files from local file system that is good enough for most of the cases. But if you are expecting a class at the runtime or from FTP server or via third party web service at the time of loading the class then you have to extend the existing class loader. For example, AppletViewers load the classes from remote web server.

### How does Java ClassLoader Work?

When JVM requests for a class, it invokes loadClass function of the ClassLoader by passing the fully classified name of the Class.

loadClass function calls for findLoadedClass() method to check that the class has been already loaded or not. It’s required to avoid loading the class multiple times.

If the Class is not already loaded then it will delegate the request to parent ClassLoader to load the class.

If the parent ClassLoader is not finding the Class then it will invoke findClass() method to look for the classes in the file system.

### Java Custom ClassLoader

We will create our own ClassLoader by extending ClassLoader class and overriding loadClass(String name) method. If the name will start from com.journaldev i.e our sample classes package then we will load it using our own class loader or else we will invoke the parent ClassLoader loadClass() method to load the class

|  |  |
| --- | --- |
| 1 | Write a program to fetch highest salary from a list of empoyees using streams |
| 2 | I have flat files coming from different source system and each source system has different file , now I need to read files and process it and save to database as per filesystem. How will I achieve the same using any of Integration framework. |
| 3 | Implement Producer Consumer design Pattern using Java 8 |
| 4 | Write a program to return key from a hashmap based on the value provided |
| 5 | Write a program passing lambda as method argument |
| 6 | Write a program to calculate area of a Shape using SOLID Design principle |
| 7 | Write a program to allow one duplicate in Set for a custom object |
| 8 | Ping pong ball program in multithreading - one thread will print Ping and other will print Pong simultaneously &  continuously |